IN THE SPECIFICATION:

Please amend the paragraph beginning on Page 8, line 30 to Page 9, line 19 as follows:

--Figures 2 and 3 show a system using an embodiment of this invention. In the preferred embodiment, the compilation activity is broken up into two phases, described in Figures 2 and 3. Referring now to Figure 2, the computer program (200) is processed by a quasi-static image generator compiler (201), referred to as *QSI writer*. The QSI writer produces one or more quasi-static images (202), referred to as *QSI*=s, which are persistent images of the executable code. The QSI=s are stored for subsequent use by the virtual machine using a QSI repository system (203). Referring to Figure 3, the computer program (304), in the form of source code or intermediate language code, such as *bytecode* in a Java virtual machine environment, is processed at run-time by a quasi-static run-time compiler (305), referred to as *QSRT compiler*. The QSRT compiler uses the QSI repository system (306) (which is identical to 203 in Figure 2) to retrieve the QSI=s (307) containing executable codes for various components of the program. After processing the QSI, the QSRT compiler generates executable code (308) that is used by the run-time system (309) for executing the program.--

Please amend the paragraph beginning on Page 17, line 16 to Page 18, line 7 as follows:

--How this mapping is used may be illustrated with an example from a Java virtual machine. The location in which a classfile is stored in a Java virtual machine can be viewed as having two components: the *repository* containing the class, and the directory structure implied by the *fully qualified name* of the class [8]. For example, a class MyPackage.Foo, appearing in a repository /vol/jdk/classes on an AIX platform, is stored in the directory

/vol/jdk/classes/MyPackage. The repository containing a class is identified by its defining class loader (e.g., using a search based on the classpath environment variable). For each class loader, a fixed mapping is defined from the name of the repository holding the class to the repository holding the QSI file, should it exist. Consider a class loader that loads classes over the network. The preferred method would use a local repository for the QSI files. Within a repository, the method uses the same directory structure for a QSI file as that implied by the fully qualified name of the class. In the above example (for the class Foo in /vol/jdk/classes/MyPackage), given a QSI repository mapping function that replaces the string ``classes" by the string ``qsi", the corresponding QSI will be stored as Foo.qsi in the directory /vol/jdk/qsi/MyPackage.--

Please amend the paragraph beginning on Page 21, line 21 to Page 22, line 15 as follows:

--Now described are further details of step 907 from Figure 10 to adapt the code and auxiliary information for a procedure to the new execution context. Figure 12 shows a pseudocode for this step. The loop 1201 goes over every item recorded in the list of adaptation annotations. Step 1202 locates the instruction in the code or the item in the auxiliary information for code, such as exception table or garbage collection maps, which is to be modified. Step 1203 locates the old information that is no longer relevant in the new execution context, and replaces it by new information computed using the symbolic reference S and information available to the virtual machine about the new execution context. If this step fails, 1204 exits the adaptation procedure with an indication of failure. If step 1203 involved adding extra instructions to the code, step 1205 updates auxiliary information for the code, such as garbage collection maps and exception table, if necessary. For example, in the context of a Java

virtual machine, if the original instruction is in a *try* block, the new instructions inserted as a replacement of the original instruction are also in that *try* block, thus we need to modify or add an entry in the exception table. To illustrate this procedure, once again we use an example from a Java Virtual Machine implementation for the IBM PowerPC architecture, discussed earlier while explaining the process of generating adaptation annotations.--